The Manic Depression of Microprocessor Debug

Don Douglas Josephson
Hewlett-Packard Company, Fort Collins, CO

Introduction

This paper describes the sometimes exhilarating, sometimes depressing, and always challenging job of bleeding-edge microprocessor debug. It covers an overview of processor debug, the flows, tools and methods used to perform debug, the process of "root causing" and fixing bugs, and includes case studies of actual bugs from recent processors. In conclusion, some of the future challenges for microprocessor testing are presented.

Finding the Needles in the Haystack

Processor debug begins once first silicon is received after the initial design is completed. It can be separated into two major components – mania and depression, which alternate in a seemingly endless loop throughout the debug of the design! When initial silicon or a revision is first tested, depression begins to set in as the reality of the extent of design bugs becomes apparent. However, as the debug process proceeds, and more knowledge is gained about the failures, depression is replaced by mania as the search for the "root cause" of the bugs is pursued with zeal, and fixes are implemented. This cycle repeats until the design is "clean" and can be shipped to customers. A structured process is followed to "find the needles in the haystack" and fix the bugs in a design. The four main elements of this process are: characterization, debug, fix, and verification.

Doctor, It Hurts When I Do This

Failures are identified either on a tester or in a system environment during the process of characterization. Failures on a tester may show up as early/late/missing bus cycles, or incorrect addresses or data coming from the bus. System failures manifest themselves as failures during the boot-up sequence, during focused manual or random instruction testcases, or even in operating systems/applications. These bugs are sourced from characterization or from simply booting or running applications. Failures are sourced by system and tester characterization teams; their job is to stimulate the device with all available methods to find bugs in the design. Failures may be either functional (the circuit is logically incorrect and does not operate properly under any condition) or electrical (the circuit is logically correct, but does not operate during variations in operating conditions like voltage/temperature/frequency). Some electrical failures may be so severe that they appear to be functional failures on some parts. Electrical failures can also be specific to certain processing conditions (faster or slower transistors or interconnect for example). Bugs can be very visible (fail on nearly every part) or so hard to find that they may only spuriously occur on the same code sequence out of billions of cycles on one part.

The process of characterization uses the concept of the "window" of operation while characterization is underway (Figure 1).

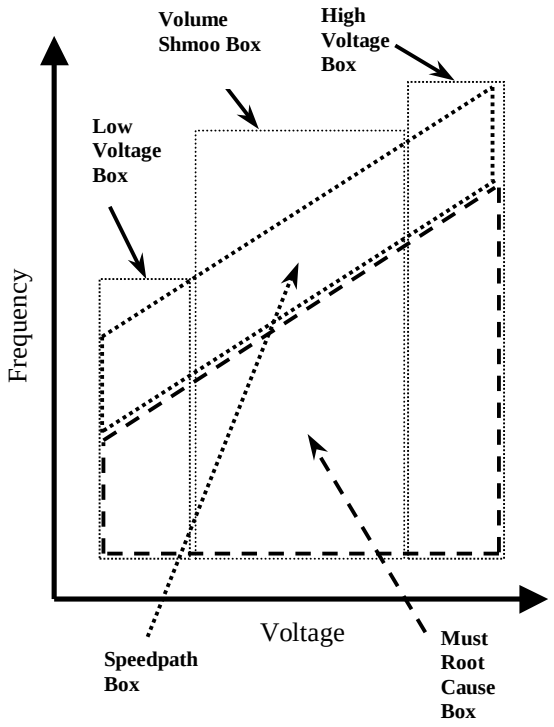

Figure 1 – Processor Characterization Window

As shown in the figure, there are several "boxes" that the characterization team cares about. Splitting the

characterization into areas allows different teams to be focused on the different possible types of failures. The first box is the volume shmoo box. This incorporates the specified frequency and voltage range, with some margin, that the processor is expected to operate over. The goal of initial characterization is to explore this window of operation and find any design problems within it. Any failure within this box is a "must fix". As characterization progresses and silicon health improves with fixes, the high voltage and low voltage boxes are explored for robustness problems in the design. The speedpath box can also be explored to find bugs that, if fixed, can improve the performance of the design by increasing the frequency of operation of the devices.

Finally, an important concept of the characterization window is the "must root cause" box. Any failures within this box must be explored and debugged such that they are completely understood. Even if the failures occur in the high and low voltage boxes outside the normal volume box, with process variation it is possible for failures in these areas to encroach in the normal operating region. Therefore, it is important to understand all of these failures, although it is possible that some may not be fixed – with root cause it is usually possible to make an assessment as to whether a particular bug could ever occur in the volume shmoo box.

Once a failure has been identified as a part of the characterization process, the process of debug begins. Failures are typically triaged (or "bucketed") into similar failure modes. The worst test case that causes the failure is found, and the problem is handed off to a debug team, which may consist of one or more engineers. Debug teams work in parallel on different failures. Debugging consists of performing experiments to determine sensitivities that a particular bug has (e.g. varying temperature, performing code experiments, trying different parts with different processing, et cetera). One of the most important techniques is "shmooing", especially when coupled with other techniques like manual and automated code experiments.

Shmoo Will Be Your Friend

The ubiquitous "shmoo" [3,4] is the debugger's best friend for electrical failures. Often it is the source of the initial investigation into failure modes. Shmoos can be performed in either systems or on testers for microprocessors. Typically, frequency and voltage are varied in order to expose overall sensitivities in the design, although on testers additional shmoos of things like I/O voltage levels relative to each other can also be very instructive in debugging failures in the external interfaces to the processor.

Shmoos run at both hot and cold temperatures can also give great insight into failures. For example, shmoos that exhibit failures at high temperature, but not cold temperatures, may indicate problems with leakage (since leakage increases), noise (since threshold voltages decrease which makes noise-sensitive circuits easier to spuriously trip), or speed (since FETs slow down).

Shmoos that fail at cold temperatures, but not at high temperatures, may indicate problems with races (since FETs and thus circuits speed up), noise (since edge rates decrease and dynamic currents increase, thus increasing di/dt and dv/dt for inductive and capacitive coupling), or charge sharing (since FETs speed up and charge is more rapidly transferred between sensitive nodes).

There are numerous different characteristic shmoos that are typically encountered during processor debug. These often vary considerably from the traditional "speedpath" shmoo encountered in CMOS designs. Debug teams often assign euphemistic names to each characteristic shmoo during debug that describe the relative characteristics of each shmoo. Figure 2 shows some examples of commonly encountered shmoo plots in the process of microprocessor debug.

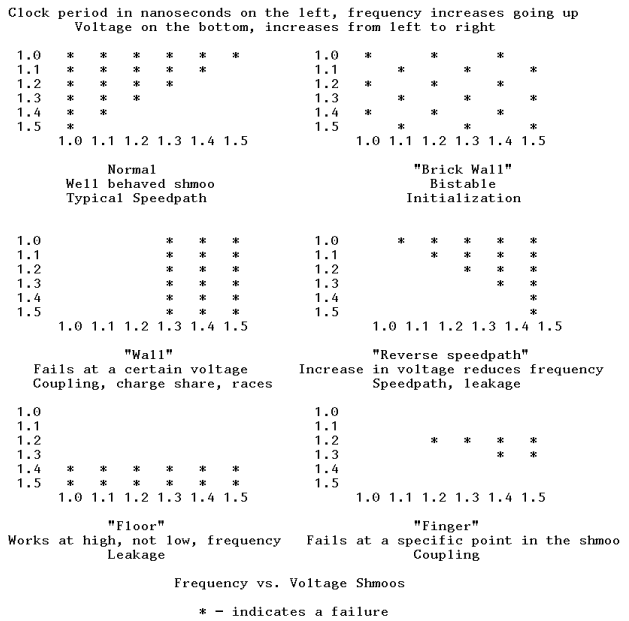

Figure 2 - Typical Processor Shmoo Plots

The "normal" shmoo is the expected behavior for a robust design. This shmoo exhibits the typical frequency

vs. voltage rolloff encountered in CMOS circuits – as frequency is increased, circuits begin failing due to speed problems. Decreasing voltage also causes more failures to occur. If only all designs were this well behaved!

Alas, for the unfortunate microprocessor debug engineer, there are many other failure mechanisms that can occur in microprocessor designs. This is usually the result of the endless creativity of the design team in concocting new and different circuit design techniques (static, dynamic, self-timed, self-resetting, asynchronous, pseudo-NMOS, pseudo-dynamic, annihilation, et cetera) in the process of solving (?) design problems.

The "brick wall" shmoo is indicative of an initialization problem, where a state element in the design is not properly initialized and alternates between a good and bad value after reset of the device. This behavior is shown in the plot, where the failure alternates from point to point.

A very common problem is the "voltage wall". This is where the device fails to operate above or below a certain voltage. Typical failure modes in circuits which result in this behavior are noise (either inductive or capacitive coupling between interconnect lines on the chip), charge sharing, or a race between sequential elements. Such behavior can also vary considerably with processing, which can cause the characterization window of the device to be severely limited early in debugging.

The "floor" is indicative of a circuit that is experiencing problems with leakage (either subthreshold or gate leakage). As the frequency is reduced, more time is allowed for charge to "leak" off of storage nodes in state elements. These failures may occur even if nodes are restored via feedback if the leakage is severe enough and the circuit is improperly designed for robustness.

Finally, the aptly named "finger" indicates a problem with coupling between metal lines that is specifically dependent on the alignment of the victim and culprit signals. At certain frequencies, culprit signals align with a victim signal in such a way so that they can cause a failure in a circuit.

Break Out the Swiss Army Knife

There are a large number of tools available to assist in the debug of processors. Some of these are embedded in the hardware of the design itself. Others allow the device to be measured or manipulated. A combination of tools is useful in debugging most failures.

One of the most important features available for debugging allows observability into the device by means of scanpaths that access internal state elements in the design. These scanpaths access device state either destructively or non-destructively [1]. Other implementations also allow for on-the-fly generation of a signature via a MISR [6]; such signatures can be compared to the passing value to determine if a failure has occurred or not at a particular operating point. Passing data can be compared to failing data between parts or the same part at different environmental conditions to narrow down where a failure may be occurring. Scan data may be captured either in systems or on testers during the process of debug.

Another very useful feature is the ability to adjust clocks internal to the device. This includes the ability to globally skew the main clock edges (to vary duty cycle and “stretch” or “compress” certain phases of the clock) as well as the ability to skew regional clocks relative to each other (between major blocks on the chip). When such adjustments are performed in a systematic way, it is possible to find a particular cycle on which a failure occurs. At that point, scan may be used to look at internal state information to see how it varies between the failure and passing cases.

In some cases, not enough scan information is available to definitively determine what the failure might be. In these cases, techniques like LVP [5] and PICA [4] become very important. These allow actual signal waveforms internal to the device to be measured with high timing accuracy in a non-invasive manner, and are useful in finding spurious circuit failures like glitches and noise. Such techniques are useful only after narrowing the failure down to a specific area of circuitry however and are not generically applicable; scan data and failing clock cycle information are required to guide such analysis.

Finally, another excellent tool for bug investigation (as well as resolution) is a focused ion beam (FIB) edit. If a failure hypothesis remains elusive, a FIB edit that affects the suspected circuit can often shed light on a failure mechanism. Also, once a failure is found, a FIB edit that works around the bug can provide definitive proof that the failure mode is understood and that the fix will work when mask edits are made.

Things that Make You Go “Hmmm…”

By combining the results of shmoo and code experiments from the tester and system, and using debug tools like

scan, clock manipulation, and LVP, it is usually possible to come up with a "root cause hypothesis" for the failure mechanism. With good scan data it is frequently possible to pin a failure right down to the problem circuit. For less observable areas like memory arrays, code experiments are important for determining what the sensitivity is (for example - is it a read or a write that fails; is it in a single bit or across a row or column; is it temperature sensitive?) Often some sort of circuit probing with LVP or PICA may need to be done to investigate failures that cannot be isolated easily by scan.

With a root cause hypothesis for the failing circuit, it is time to find the "light switch test". With true root cause for a bug, a debug engineer should be able to turn the failure "on and off" by varying a condition. This might be simply changing a code sequence or a data pattern, or in a complex case may involve actually doing a FIB edit to prove the circuit sensitivity by adjusting a FET size or changing another circuit parameter. Hopefully, the hypothesis is correct, and the light switch test works. If not, the process of experimentation begins again based on the data collected previously.

One Rat – Many Rats

Once the definitive root cause for a failure is determined, the hunt is on for any similar problems in the design – i.e. if you see one rat, there are probably many more that you can't see! Therefore, an exhaustive search is undertaken for similar possible bugs in the design. Often this is based on an automated approach, as a tool can usually be written to check for any other similar problems, and it is much faster than doing a manual search for problems.

Get Out the RAID Can

There are usually several possible fixes that can be made to fix a given bug. Often, current program conditions dictate the type and urgency of a particular fix. If a bug is "blocking" – i.e. there is no possible way to work around it, and it threatens to "block" the discovery of further bugs, it must be fixed as soon as possible. An example of this might be a race in the instruction issue circuits – a circuit failure here blocks the discovery of further bugs. Lower urgency bugs can be deferred until the next release. Examples of this might be a test feature that is not required until production testing starts, or a bug that has a software workaround.

Processor debug projects attempt to adhere to a schedule of releasing new versions of masks every so often, and this schedule is a mix of both metal revisions (changing only interconnect layers) and full turns (which change every mask layer – i.e. transistor edits can also be made). The debug process can last anywhere from 9 months to 18 months depending on the complexity of the design and whether or not it is a brand-new design or just a modification and process shrink of an existing design.

A graph for revisions required for a typical processor along with the number of bugs fixed is shown in Figure 3. Early on, it is necessary to fix a few major blocking bugs, and it can be difficult to "work around" these bugs to find others. As debug progresses, it is possible to find more bugs as the characterization window opens up, which leads to a larger number of bugs found. Also note that a large number of bugs are fixed in the full transistor-level revision 2.0; such a revision includes many bugs that can only be fixed in a full turn. The timing of particular bug fixes must be factored into the project schedule, which also takes into account the needs for customer prototypes, manufacturing test development and product qualification in addition to the needs of the debug team.

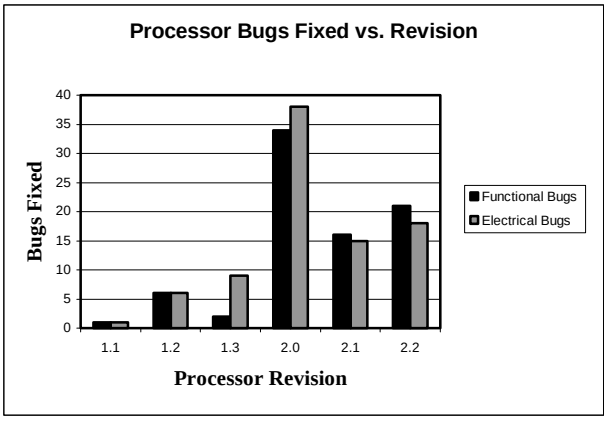

Figure 3 - Silicon Revision Bug Profile

Depending on the complexity of a fix, it may be possible to perform a FIB (focused ion beam) edit to repair the failing circuitry. This can be an attractive option for fixing a small number of parts to verify that the problem can be fixed with a later mask edit. Also, this can allow additional characterization to proceed if the bug is blocking other verification efforts. If the edit is simple, a large number of FIB edits can be performed to fix many parts to enable volume characterization to proceed. This can be important in accelerating validation before silicon can be revised with a mask change.

Debug Case Studies

There are numerous possible failure mechanisms for microprocessor bugs. Failures tend to be quite diverse due to the wide range of circuit techniques used in processor design. Figure 4 shows a breakout of failure types for electrical bugs on a typical microprocessor.

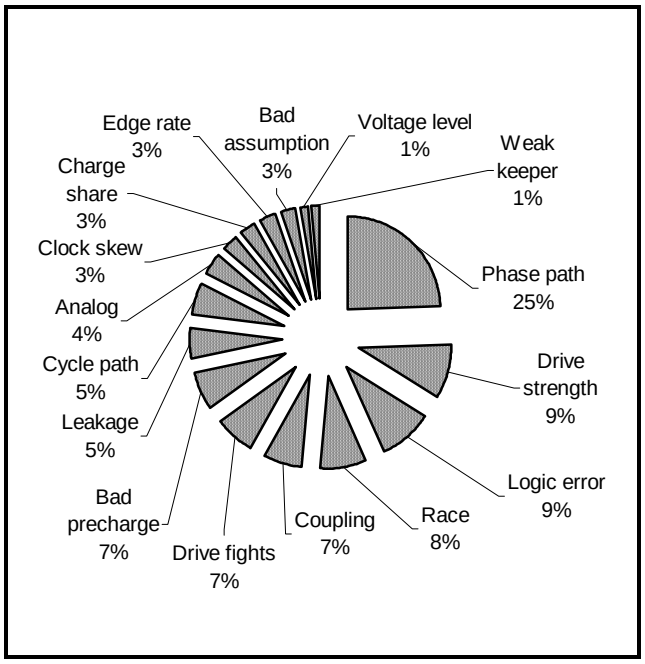

Figure 4 – Electrical Failure Mechanisms

Note that the distribution of failure mechanisms is fairly uniform, which indicates that the design methodology was generally sound (i.e. there were no major "holes" in the design methodology). Phase paths (typically dynamic circuits) were the dominant speedpath failure.

Below are descriptions of two microprocessor bugs. The processes used to find and to determine root cause of the bugs are described, along with the methods used to fix them. Additional example bugs and details will be presented at the conference.

I Didn't Know it Worked Like *That*

This bug was first detected during tester characterization of a microprocessor. When performing an architected instruction which caused the simultaneous write of a large number of RAM cells in an array, a "black shmoo" was observed on the tester, meaning that over a large voltage and frequency range, the part failed to operate correctly. Using another instruction that only wrote one of the RAM cells at a time, the circuitry worked fine. At first, it was theorized that this was a functional bug with the first instruction, because the circuit never seemed to work under any condition. However, as more parts were shmooed, it became clear that it was actually a very serious electrical problem: the processor could do the right thing, but only in a very small operating window.

The obvious difference between the instructions was that the first instruction attempted to write all cells at the same time. Upon investigation of the circuitry, it was discovered that the driver of the global bitlines going to the RAM cells was severely undersized. It was fully capable of driving and overriding several cells at once (thus enabling the second instruction to work) but could not simultaneously write all of the cells, as the suspect instruction was supposed to do.

The root cause of the bug was a mistaken assumption between the physical and control designer. The physical designer did not understand how an architected function worked under all conditions. The designer assumed that only one cell could be written at once and did not realize that an instruction existed which could write all the cells simultaneously. As a result, the circuit was designed improperly and could not set the large number of cells when required. Shmoo experiments were important in determining that the bug was electrical and not functional in nature. The bug required a full transistor turn to fix; extensive edits were made to bring a separate write path into each cell with a local driver to enable simultaneous writes of all the cells to support correct operation. This bug points out that the interaction between physical and functional designers is very important, and that proper validation of physical and logical assumptions is critical. Switch-level simulation should have caught this problem, but there was a problem with the tool that caused it to not detect this bug when a testcase was written which exercised the array. This problem was corrected as a result of discovering this bug.

But… It Worked in SPICE…

This problem was first discovered on a new revision of a processor, which had faster transistors than had been seen previously during characterization. The failure was that cache RAM array tests were failing during normal production test screening. Large numbers of parts exhibited the problem. Extensive shmoos were performed on several parts, and the worst-case conditions for failure were found to be high temperature and at high voltage. Pattern experiments were begun to attempt to

learn more about the failure. Also, maps of the failures and their spatial relationship to the physical positioning of the cache arrays were correlated, in an attempt to determine if physical location was important to the failure. It was noted that subarrays toward the edge of the die had more failures.

By writing cells at high voltage and reading them back at low voltage, it was determined that writes were robust at high voltage, and that the read was failing. Further investigation of the read circuits for the array showed that there was a potential for excessive leakage to be occurring in the dynamic global dump circuit that read the outputs of the RAM cell array. This circuit had a large number of NFET column select FETs attached to a dynamic node, which had a precharger PFET and a weak PFET keeper to help hold the node when the precharger was off and none of the NFET column selects were on.

With high voltage and high temperature failures, and based on the fact that the parts having the problem had faster (and thus leakier) transistors, it seemed likely that leakage was occurring in the cell. FIB experiments were performed to cut off some of the NFETs on the dynamic storage node, and this improved the failure markedly, which indicated that the root cause was indeed leakage.

The causes of this bug were inadequate circuit design and a variation in processing that caused excessive leakage in the NFET column select transistors. The PFET keeper methodology for dynamic nodes had been designed to fight off-NFET leakage adequately, but the keeper in this circuit did not meet methodology guidelines for sizing (in an attempt to gain speed, the keeper was downsized). As a result, the PFET keeper was too weak to hold the dynamic storage node up even when the NFETs were turned off. SPICE had been performed to show that the keeper would hold against leakage, but over the course of the project, the SPICE models changed, with a resulting 2X increase in leakage current (which matched silicon), but the keeper simulation was not rerun with the new models which reflected actual silicon performance.

In addition to the inadequate keeper size, another problem was that the NFETs in the evaluation tree of the dynamic circuit were excessively leaky due to a peculiar layout interaction, which resulted in the actual FETs "narrowing" in the center. This increased the leakage of the transistors even more, exacerbating the failure. Such interactions between the design and process can make debug difficult, as many parts need to be examined to see the process variation which can affect circuit behavior.

Future Challenges for Microprocessor Debug

As if validating current microprocessors wasn't already hard enough, there are a number of emerging challenges for debugging these devices in the future.

Perhaps one of the most important issues facing microprocessor designers today is controlling power. With decreasing transistor geometries and higher speeds, microprocessors are becoming ravenous consumers of power. Gate leakage is becoming a major factor in leading-edge processes as well. Ever-increasing power consumption poses a number of problems for processor debug. One is the issue of adequately powering the device under conditions in which it normally does not operate (high voltage and frequency) in order to perform adequate electrical characterization to ensure operating margin. This problem is especially acute in systems, where regulator modules can only supply a limited amount of power, but it is also a problem on ATE, where large di/dt transients can pose problems for power supplies. Inadequate power supply to the processor can lead to "brown out" and the inadvertent debug of what appear to be circuit problems which are simply related to inadequate power delivery.

Another complication will arise when processor designs begin to take advantage of throttling voltage and frequency as a method of controlling power in normal operation. Dynamic adjustments in voltage and frequency could pose difficult challenges in debugging processors in the future, due to the non-deterministic nature of these adjustments. Cooling devices for probing techniques like PICA and LVP will also be a difficult challenge.

Another emerging trend is the increasing demand for bandwidth into processors from memory to satisfy very high CPU core clock rates (in the GHz range). The need to reduce system cost for multiprocessor systems makes "glueless MP" very desirable. Such interfaces are often designed using high frequency serial links, with transfer rates in the GT/s range. These interfaces are typically asynchronous relative to the processor core logic. The asynchronous nature of these interfaces can pose severe difficulties in reproducing failures in a deterministic manner when debugging a processor.

In addition, the very high transfer rates of these interfaces make attaching traditional debug tools like logic analyzers nearly impossible. Such difficulties indicate a need for incorporating LA functionality directly into processors. High data rates, coupled with

wide interfaces, also seem to be spelling the end of high speed functional testing using automated test equipment. Processor data transfer rates and pin requirements are increasing far more rapidly than the capability of testers to keep up is. It is simply not going to be cost-effective for companies to buy at-speed testers that can support such interfaces, if such testers will even exist in the future. Structural test methods, coupled with extensive on-chip circuitry to support high-speed interface verification, seem to be the only answer to this problem.

Another challenge in processor debug is process variation and its effect on circuits. As even smaller feature sizes are used in future processes, FET variation in designs with billions of transistors will be a growing problem. With a large number of transistors in a device, designers are guaranteed to get some transistors that lie at the edge of the process variation window. Random variation in transistors may preclude the use of "tricky" circuits in processors in many cases, and designers in the future may have to fall back on more robust design techniques. Such designs are also far more scalable into future processes, and require much less characterization effort, thus reducing time to market.

Lithography issues like optical proximity correction and large reticle size can also introduce unanticipated effects in circuit performance. In the future, these also may need to be accounted for in design to ensure adequate yield, even when circuits meet design rule checks. Consider the image shown in Figure 5, of several FETs in a cell. The cell is mirrored about the Y-axis in the center of the picture. Note that while the FET on the left is fabricated correctly, the supposedly identical FET on the right is half of the proper length! Is this a defect, or normal process variation? Will processor circuits of the future need to cope with this sort of extreme variation?

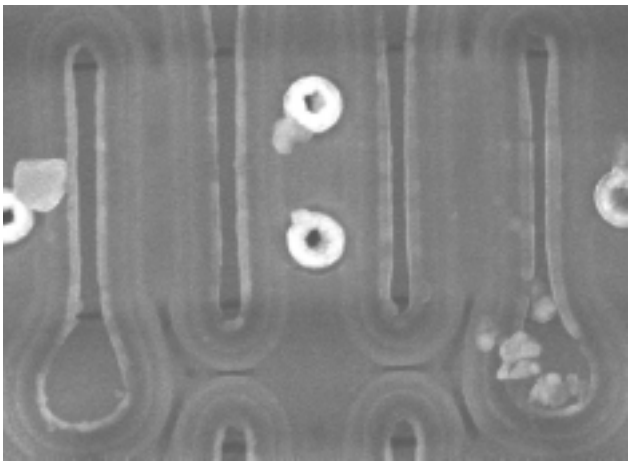

Figure 5 – Failure Analysis Picture

Only time will tell which of these challenges will materialize as being major limiters in processor debug, but one thing is certain – such issues as well as other unforeseen ones will continue to challenge microprocessor design and debug teams in the future!

References

[1] D. Josephson, et al., "Debug Methodology for the McKinley Processor", Proceedings of the International Test Conference, 2001.

[2] J. Bockhaus, "Electrical Verification of the HP PA8000 Processor", Hewlett-Packard Journal, August 1997, pp. 32-39.

[3] K. Baker, J. V. Beers, "Shmoo Plotting: The Black Art of IC Testing", IEEE Design and Test of Computers, July/September, 1997, Volume 14, Number 3, pp. 90-97.

[4] W. Huott, et al., "The Attack of the 'Holey Shmoos': A Case Study of Advanced DFD and Picosecond Imaging Analysis (PICA)", Proceedings of the International Test Conference, 1999, pp. 883-891.

[5] M. Paniccia, et al., "Novel Optical Probing Technique for Flip Chip Packaged Microprocessors", Proceedings of the International Test Conference, 1998, pp. 740-747.

[6] A. Carbine, D. Feltham, "Pentium® Pro Processor Design for Test and Debug", Proceedings of the International Test Conference, 1997, pp. 298-299.